Cubic Splines

Marina Peñalver Ripoll

Contents

Introduction

A **cubic spline regression** is defined as a piecewise differentiable curve. A distinctive feature of splines is that, although they are composed of separate segments, the resulting function is smooth across the entire domain, including at the joints, known as **knots**.

Cubic splines use third-degree polynomials in each segment, ensuring not only the continuity of the function but also the continuity of its first and second derivatives across the domain.

Knots are crucial as they connect the spline segments. Typically, knots are evenly distributed over the range of the independent variable or positioned at percentiles of the variable's distribution. The locations of the knots are denoted by:

$$\{x_i^* : i = 1, \ldots, k\}. \tag{1}$$

The most intuitive way to construct a basis for a cubic spline starts with a cubic polynomial basis: 1, x, x^2, x^3, and adds a truncated power basis function for each knot. For each knot x^*, the truncated power function is defined as:

$$h(x, x^*) = \begin{cases} (x - x^*)^3 & \text{if } x > x^* \\ 0 & \text{if } x \le x^* \end{cases}.$$

The resulting function basis for a cubic spline is:

$$\begin{aligned} & b_0(x) = 1, \\ & b_1(x) = x, \\ & b_2(x) = x^2, \\ & b_3(x) = x^3, \\ & b_{i+3}(x) = h(x, x_i^*), \quad \text{for } 1 \le i \le k. \end{aligned}$$

In this case, the basis consists of $q = k + 4$ elements. Fitting a cubic spline to data is formulated as a linear regression model, similar to the one described earlier. The resulting model is $y = X\beta + \varepsilon$, where the i-th row of X is given by:

$$X_i = \left[1, x_i, x_i^2, x_i^3, h(x_i, x_1^*), \ldots, h(x_i, x_k^*)\right].$$

Simulating data

For our examples, we will simulate data to explore how well the advantages of cubic splines.

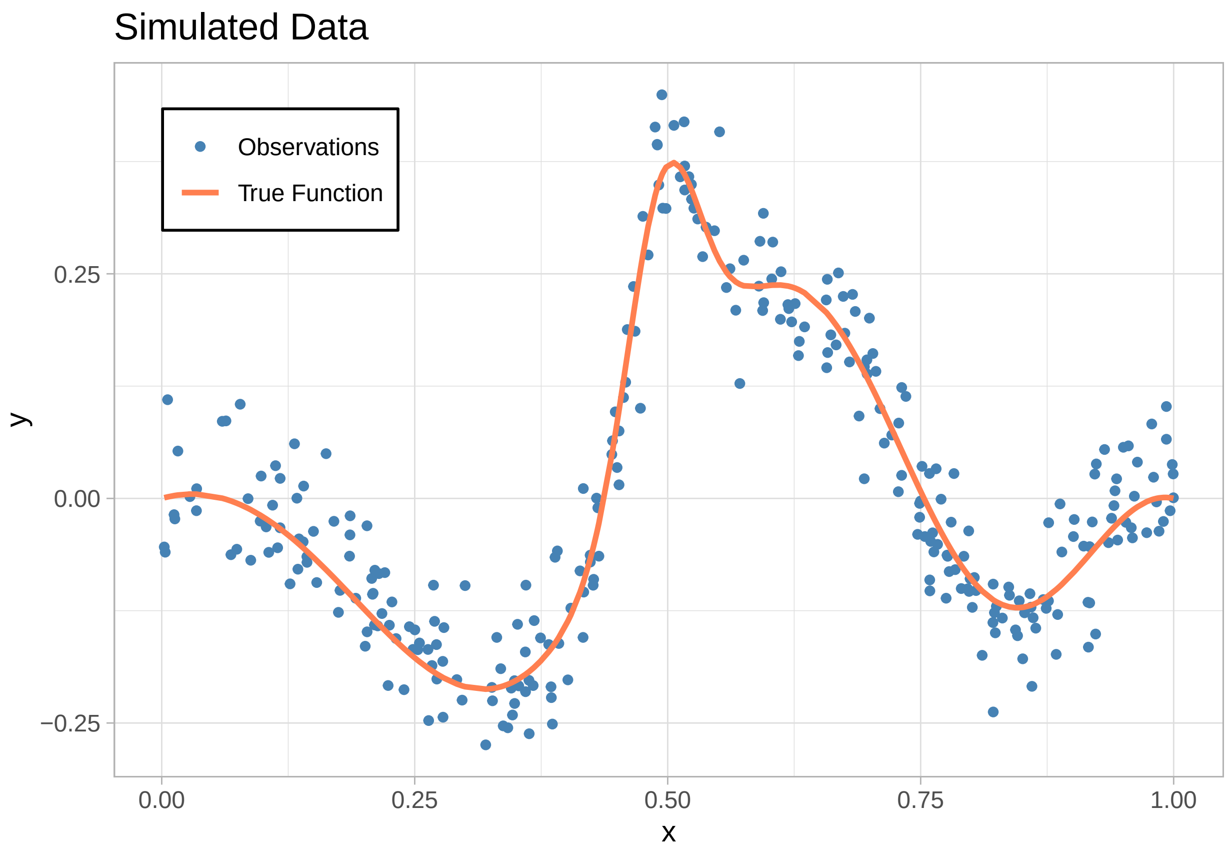
```r
# Generate X values
x <- runif(300, min=0, max=1)
x <- x[order(x)]

# Error term
eps <- rnorm(300, mean=0, sd=0.05)

# True and observed Y values
y.true <- -x * cos(5 * (x + 0.5)^2) * (x - 1) +
          1.2 * exp(-(20^2) * (x - 0.5)^2) * (x - 1)^2
y.obs <- y.true + eps
```

To visualize the data, we utilize ggplot2:

```r
library(ggplot2)
```

```r
data <- data.frame(x, y.obs, y.true)
ggplot(data = data) +
  geom_point(aes(x = x, y = y.obs, color = "Observations"),
             shape = 20, size = 2) +
  geom_path(aes(x = x, y = y.true, color = "True Function"), linewidth = 1) +
  scale_color_manual(values = c("Observations" = "steelblue",
                                "True Function" = "coral")) +
  labs(title = "Simulated Data",
       x = "x",
       y = "y",
       color = NULL) +
  theme_light() +
  theme(plot.title = element_text(size = 14),
        legend.position = c(0.15, 0.85),
        legend.background = element_rect(fill = "white",colour = "black"))
```

```
## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2
## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



`bs` Function

To fit splines, it is necessary to load the `splines` package in R. We will use the `bs` function for this purpose.

```r
library(splines)
```

The `bs` function is used to generate bases for spline regression models. The abbreviation `bs` stands for "basis spline." This function is particularly useful for flexibly modeling nonlinear relationships between variables. The general syntax of `bs` is:

```r
bs(x, df = NULL, knots = NULL, degree = 3, intercept = FALSE, Boundary.knots = NULL)
```

The key parameters are:

- `x`: The independent variable, the data on which the spline will be applied.
- `df`: Degrees of freedom for the spline. This parameter controls the total number of terms in the model, which is the number of internal knots plus the degree of the polynomial plus one.
- `knots`: A vector of points within the range of x that specifies the internal knots of the spline. If not specified, knots are placed at quantiles of the data that correspond to even spacing in terms of the distribution of x.
- `degree`: The degree of the polynomials used. The default is 3, which corresponds to a cubic spline.
- `intercept`: A boolean indicating whether an intercept term should be included in the spline basis. The default is `FALSE`.
- `Boundary.knots`: A vector of length two that specifies the knots at the boundaries of the data range. If not specified, the boundary knots are the minimum and maximum values of x.

In our analysis, if we let k represent the number of internal knots, then the relationship between k, the degrees of freedom (df), and the degree of the spline can be expressed as:

$$k = df - (degree + 1), \quad \text{or alternatively} \quad df = k + degree + 1.$$

This formulation assumes that an intercept is included in the model. If the intercept is not included, the formula adjusts to:

$$df = k + degree.$$

Let's start by exploring the function. We'll set up a cubic spline with 2 internal knots:

```r
degree <- 3
k <- 2

# The location of the knots is
attr(bs(x, df = k + degree), "knots")
```

```
## [1] 0.3797453 0.7280820
```

It's also possible to specify the knots manually:

```r
attr(bs(x, knots=c(0.1, 0.3, 0.4, 0.5, 0.55, 0.7, 0.9)), "knots")
```

```
## [1] 0.10 0.30 0.40 0.50 0.55 0.70 0.90
```

Fitting the Data

We will start our analysis with a cubic spline of 10 knots.

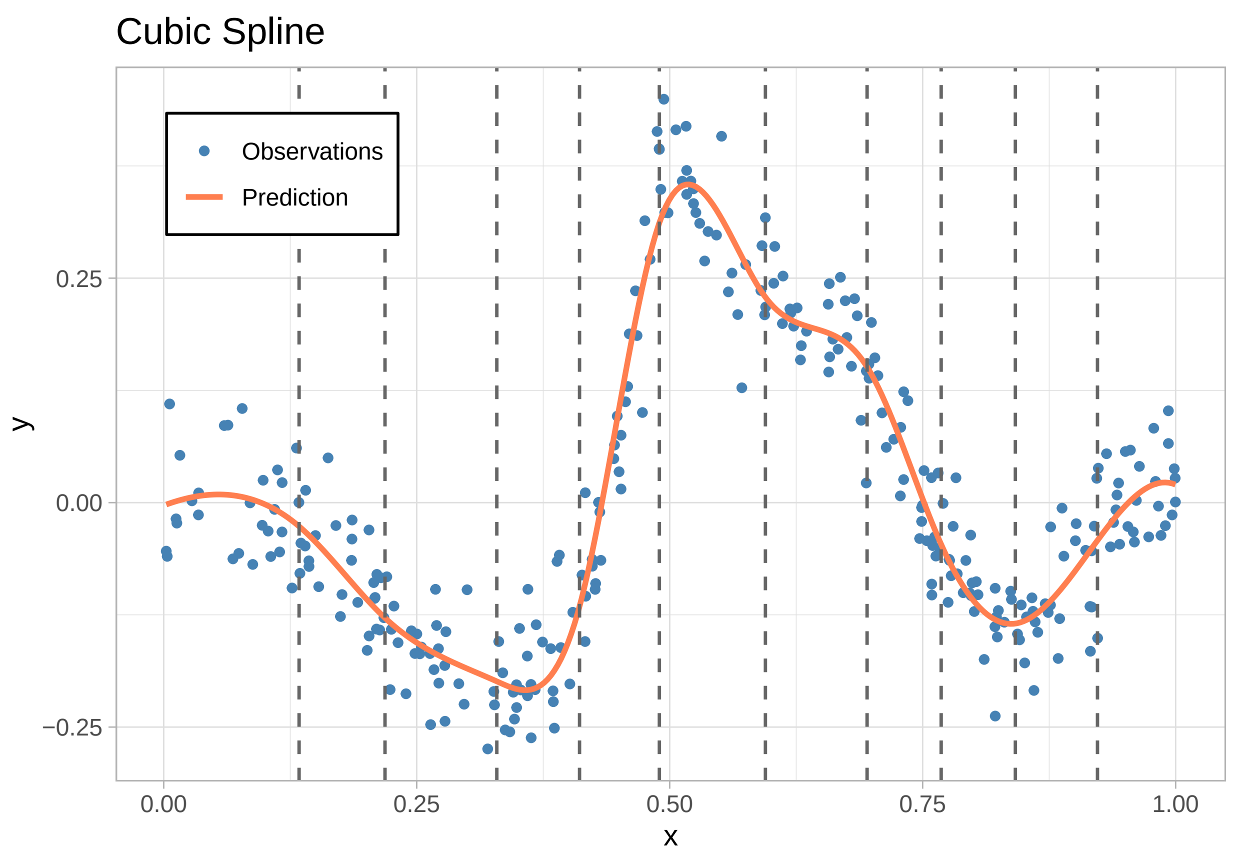
```r
degree <- 3
k <- 10
spline.model <- lm(y.obs ~ bs(x, df=k + degree, degree=degree))
```

We can compute predictions as usual:

```r
x.pred <- seq(min(x), max(x), length.out = 300)
y.pred <- predict(spline.model, newdata = data.frame(x = x.pred))
```

Next, the results are shown:

```r
data.spline <- data.frame(x.pred, y.pred)
knots <- attr(bs(x, df = k + degree), "knots")
ggplot(data, aes(x = x, y = y.obs)) +
  geom_point(aes(color = "Observations"), shape = 20, size = 2) +
  geom_line(data = data.spline, aes(x = x.pred, y = y.pred, color = "Prediction"),
            linewidth = 1) +
  geom_vline(xintercept = knots, linetype = "dashed", color = "gray40",
             linewidth = 0.6) +
  scale_color_manual(values = c("Observations" = "steelblue",
                                "Prediction" = "coral")) +
  labs(title = "Cubic Spline", x = "x", y = "y", colour=NULL) +
  theme_light() +
  theme(plot.title = element_text(size = 14),
        legend.position = c(0.15, 0.85),
        legend.background = element_rect(fill = "white", colour = "black"))
```



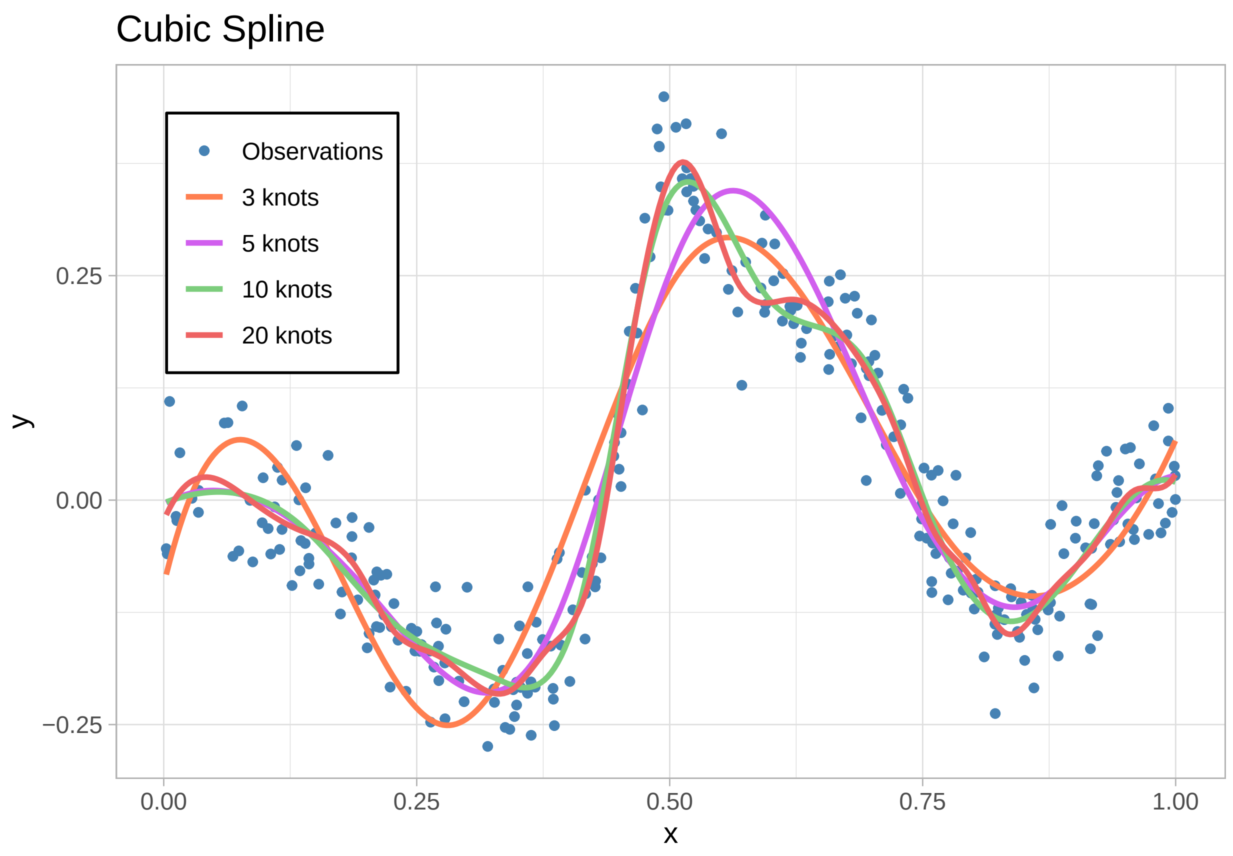
```r
# ggsave("../graphs/CubicSpline1.pdf", width = 10, height = 6, dpi = 300)
```

Now, let's compare how the number of knots affects the spline fit:

```r
degree <- 3

# Models
spline1 <- lm(y.obs ~ bs(x, df=3 + degree, degree=degree), data = data)
spline2 <- lm(y.obs ~ bs(x, df=5 + degree, degree=degree), data = data)
spline3 <- lm(y.obs ~ bs(x, df=10 + degree, degree=degree), data = data)
spline4 <- lm(y.obs ~ bs(x, df=20 + degree, degree=degree), data = data)


# Predictions
x.pred <- seq(min(x), max(x), length.out = 300)
y.pred1 <- predict(spline1, newdata = data.frame(x = x.pred))
y.pred2 <- predict(spline2, newdata = data.frame(x = x.pred))
y.pred3 <- predict(spline3, newdata = data.frame(x = x.pred))
y.pred4 <- predict(spline4, newdata = data.frame(x = x.pred))
```

For the visualization:

```r
data.spline <- data.frame(x.pred, y.pred1, y.pred2, y.pred3, y.pred4)

ggplot(data, aes(x = x, y = y.obs)) +
  geom_point(aes(color = "Observations"), shape = 20, size = 2) +
  geom_line(data = data.spline, aes(x = x.pred, y = y.pred1, color = "3 knots"),
            linewidth = 1) +
  geom_line(data = data.spline, aes(x = x.pred, y = y.pred2, color = "5 knots"),
            linewidth = 1) +
  geom_line(data = data.spline, aes(x = x.pred, y = y.pred3, color = "10 knots"),
            linewidth = 1) +
  geom_line(data = data.spline, aes(x = x.pred, y = y.pred4, color = "20 knots"),
            linewidth = 1) +
  scale_color_manual(values = c("Observations"="steelblue",
                                 "3 knots"="coral",
                                 "5 knots"="mediumorchid2",
                                 "10 knots"="palegreen3",
                                 "20 knots"="indianred2"),
                     limits = c("Observations", "3 knots", "5 knots",
                                 "10 knots", "20 knots")) +
  labs(title = "Cubic Spline", x = "x", y = "y", colour=NULL) +
  theme_light() +
  theme(plot.title = element_text(size = 14),
        legend.position = c(0.15, 0.75),
        legend.background = element_rect(fill = "white", colour = "black"))
```



```r
# ggsave("../graphs/CubicSpline2.pdf", width = 10, height = 6, dpi = 300)
```

It is observed that increasing the number of knots also increases the flexibility of the adjustment. For example, in the interval $[0.5, 0.7]$ there is a curve that the models with 3 and 5 knots fail to capture, while the model with 20 knots does capture it. Increasing the number of nodes also causes an increase in model complexity, so it is important to carefully manage this complexity to avoid overfitting.

Natural Splines

Unfortunately, cubic splines can exhibit high variance at the extremes of the predictor range, that is, when the variable x takes on very low or very high values. Natural cubic splines provide a solution to this issue by imposing constraints at the domain boundaries.

A **natural cubic spline** is constrained to behave linearly at the boundary; specifically, in regions where x is less than the smallest knot or greater than the largest knot. These additional constraints generally result in more stable estimates at the extremes of the data range, making natural cubic splines a preferred choice for modeling when data extends towards the edges of the domain.

In this case, the function we will use is `ns` from the same package.

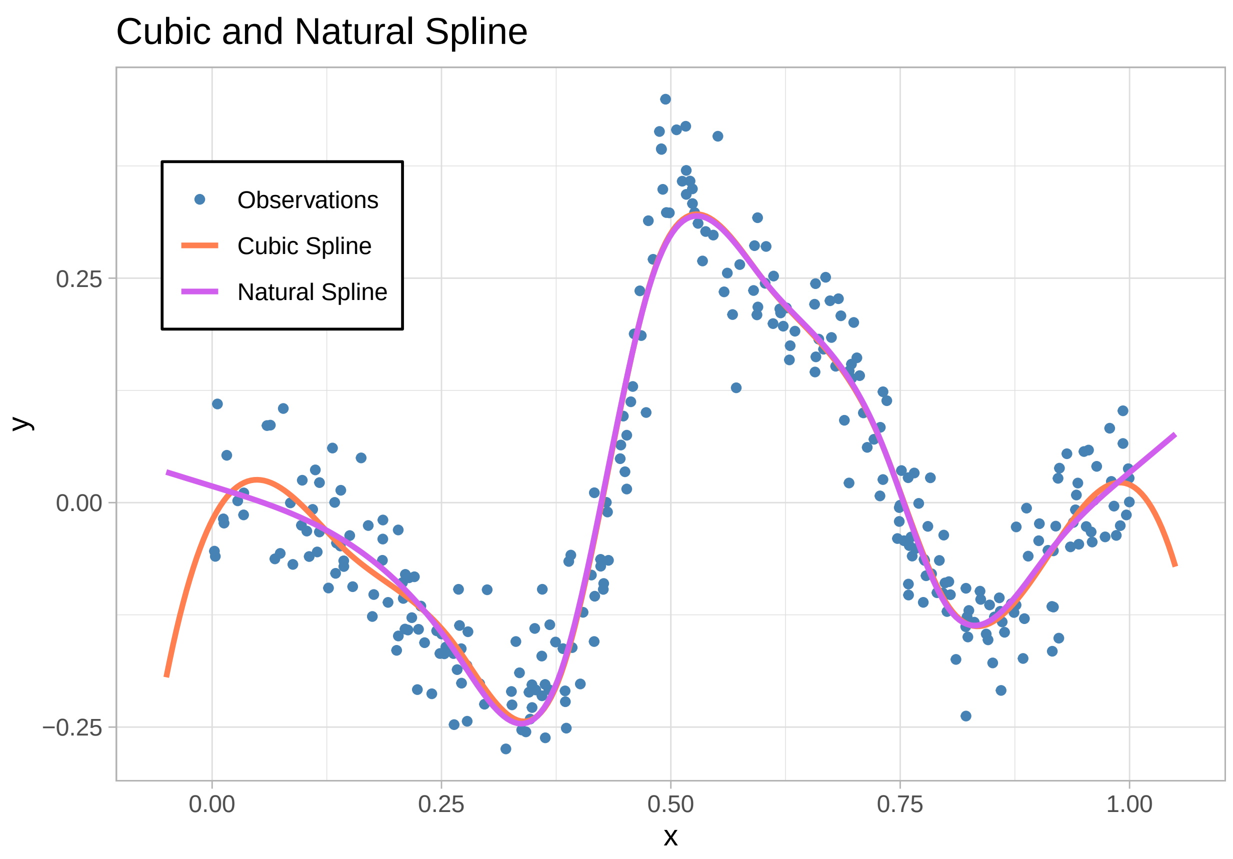
```r
degree <- 3
K <- 8

# Models
splineC <- lm(y.obs ~ bs(x, df=K + degree, degree=degree))
splineN <- lm(y.obs ~ ns(x, df=K + degree-2))

# Predictions
x.pred <- seq(-0.05, 1.05, length.out = 300)
y.predC <- predict(splineC, newdata = data.frame(x = x.pred))
```

```
## Warning in bs(x, degree = 3L, knots = c(0.144992507761344, 0.263414425413228, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```r
y.predN <- predict(splineN, newdata = data.frame(x = x.pred))
```

The results are:

```r
data.spline <- data.frame(x, y.obs ,x.pred, y.predC, y.predN)

ggplot(data.spline, aes(x = x, y = y.obs)) +
  geom_point(aes(color = "Observations"), shape = 20, size = 2) +
  geom_line(data = data.spline, aes(x = x.pred, y = y.predC,
                                    color = "Cubic Spline"), linewidth = 1) +
  geom_line(data = data.spline, aes(x = x.pred, y = y.predN,
                                    color = "Natural Spline"), linewidth = 1) +
  scale_color_manual(values = c("Observations"="steelblue",
                                "Cubic Spline"="coral",
                                "Natural Spline"="mediumorchid2"),
                     limits = c("Observations",
                                "Cubic Spline",
                                "Natural Spline")) +

  labs(title = "Cubic and Natural Spline", x = "x", y = "y", colour=NULL) +
  theme_light() +
  theme(plot.title = element_text(size = 14),
        legend.position = c(0.15, 0.75),
        legend.background = element_rect(fill = "white", colour = "black"))
```



```
# ggsave("../graphs/CubicSpline3.pdf", width = 10, height = 6, dpi = 300)
```

At the extremes of the interval, the linear trend of the natural spline is observed, while the cubic spline shows a more volatile behavior, exhibiting significant oscillations that may extend beyond the available data. This behavior is because cubic splines place no constraints on the derivatives at the end of the data, often resulting in unrealistic extrapolations and overfitting in these areas.